TU Clausthal

Clausthal University of Technology

Maintaining Sequences of Knowledge Bases in ASP

J. C. Acosta Guadarrama

IfI Technical Report Series IfI-09-11

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Michael Köster
Contact: michael.koester@tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Maintaining Sequences of Knowledge Bases in ASP

J. C. Acosta Guadarrama

Institute of Computer Science,
TU-Clausthal, Germany
jguadarrama@gmail.com

Abstract

Updates of knowledge bases have become an important topic in Artificial Intelligence and a key problem in knowledge representation and reasoning. One of the latest ideas to update logic programs is choosing between models of generalised answer sets to overcome disadvantages of previous approaches. This paper is an extended proposal of a semantics for updates that performs the methods presented in previous reports of relaxing an original knowledge base and of establishing preferences amongst candidate models. In particular, the intuition behind the methodology consists in favouring the latest updates that conflict with previous knowledge bases in a sequence, by means of *preferring generalised answer sets* and by preserving *consistency*. The semantics presented in this paper satisfies a set of *structural properties for sequences* to update logic programs, as well as other needed properties of *consistency preservation* and *inconsistent updates*. Besides satisfying such principles, this paper illustrates, by means of several examples and transformations, how to overcome problems occurring in alternative update approaches for ASP. In particular, one of the key transformations yields an abductive program out of a given sequence of updating extended logic programs. Another transformation constructs a *preferred weak-constraint program* to find *generalised answer sets* of the abductive program. Finally, as one main goal of Logic Programming, this paper describes a *functional prototype* of the declarative semantics version, that finds the update answer sets of a given sequence by means of DLV's *weak-constraints models*. The prototype is fully functional and runs online with a standard browser interface. The section also includes an analysis of complexity of the corresponding processes, an outline of the basic structure of the system, as well as a description of the employed technology.

1 Introduction

A traditional and general goal of belief updates[1] is dealing with contradictory information or with new data from a changing environment. As a result, several proposals have been formulated to update sequences of logic programs [ALP+99, EFST02, ABBL04, ABBL05], and also introduced as a survey in [Gua07c]. According to these semantics, knowledge is given by a sequence of logic programs where each program is considered an update of the previous one.

All of these semantics are based on the notion of *causal rejection of rules* [ALP+99, ABBL05, EFST02], which enforces that, in case of conflicts between rules, *more recent rules override older ones* [ALP+99, EFST00, ABBL05]. However, there are particular rare challenging (even so possible) situations that might lead to *counterintuitive models* of knowledge that have been subject of recent research and matter of formulation of new principles. As a preliminary result to solve these particular situations, [OZ04, Gua08, GDOZ05] have proposed approaches based upon the *logical contents* of programs rather than the *syntactical causal rejection of rules*. Nevertheless, despite the capability to overcome counterintuitive results of *unforeseen situations*, the main drawback of the semantics pointed out in those references is the *limitation to only one update*.

In need of a correct way to represent *dynamic knowledge*, some authors like [EFST02, SI03, ZF05] made their foundations on *Answer Set Programming* [GL88] —or simply ASP— for being one of the most solid and studied semantics for logic programs up to the last decades. In addition and as an alternative, [ALP+99] proposed an approach of DyLP on *Well-founded Semantics* (WFS) [ALP+99, ABBL05], which may be less complex than ASP (see [Dix95] for a deeper comparison), but also less *representative*.

The original problem discussed earlier in [EFST02, ABBL05, GDOZ05] is that most of the mentioned approaches are founded on their *causal-rejection principle* [ALP+99, ABBL05, EFST02]that leads to *counterintuitive behaviour* under certain circumstances, like *redundancy* or *tautological updates*. In order to illustrate this claim, consider the following theory, proposed to solve a very similar problem in [ABBL05], describing a particular situation of the sky.

Example 1.1. *Suppose an agent who believes that* when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds. *Finally, that at the current moment it is a fact that there are no* stars. *This*

[1] Notice that in practice, computing researchers call it *updates* although the difference with *belief revision* is both technical and philosophical!

simple story may be coded[2] *into* Π_1 *as follows:*

$$\begin{aligned}\Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars &\leftarrow \top \}\end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$*. Later, the agent acquires new information stating that* stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2*. As soon as the agent updates* Π_1 *with program*

$$\begin{aligned}\Pi_2 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars \}\end{aligned}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*:* $constls$*. As the model of* Π_2 *is obviously the empty answer set,* $constls$ *is considered* synonym *of* $stars$ *by means of* Π_2*, and thus the update should not change the* original beliefs*. However, the update yields an* extra answer set *in many of the existing update semantics based on the* causal-rejection principle *—see [ALP*$^+$*99, EFST02, ABBL05]:* $\{stars, constls, night\}$ *which does not coincide with* common intuition*.*

The reason is that, although $stars$ *can not be true, introducing* $constls$ *gives another possibility for* $stars$ *to be true. Thus, the additional answer set is implied [GDOZ05].*

In general, these supplementary rules in the update are a conservative extension *[ONA01] to* Π_1*: the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation,* $constls$ *should be true if and only if* $stars$ *is true.*

A solution to the problem has been introduced in [Gua08, GDO06] who propose a semantics based upon *Generalised Answer Sets* [KM90, BG03] —or *GAS* hereafter, — that satisfies several structural properties, overcoming the above problems from the proposals analysed in [Gua07c].

In this extended version, the reader can find a particular case of the framework introduced in [GDO06, Gua07a, Gua08] to perform sequences of updates, that overcomes the referred disadvantages of the causal-rejection principle. In addition, this paper includes a general description to implement update sequences in Section 7, as proposed in [GDO06, Gua07a], that leads to a running prototype, as well as tools, methods, and directions to get the running solver itself, preceded by the definition of the semantics in Section 3 and properties in Section 4. Finally, the paper ends with concluding remarks in Section 8.

[2] Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent runs across this piece of information.

2 Preliminaries

This section is quite general and as a result, the reader is expected to be familiar with basic notions of logic programming and non-monotonic reasoning form the literature.

2.1 Logic Programming and Answer Sets

One foundation of this proposal is *Answer Set Programming*, ASP [GL88], *characterised* in some non-classical logics [Pea99a], with a long background and suitability to represent *non-monotonic knowledge*. Its main applications in *problem solutions* range from typical AI *toy examples* to yet-preliminary *agent prototypes* and *planning settings*. Other name to identify this semantics from the literature is *Stable Model Semantics* or simply SM for its name in the original paper [GL88].

The following formalism gives the description of ASP, which is identified with other names like *Stable Logic Programming* or *Stable Model Semantics* and A-Prolog. Its formal language and some more notation are introduced as follows.

Definition 2.1 (ASP Language of logic programs, $\mathcal{L}_{\mathsf{ASP}}$). *In the following $\mathcal{L}_{ASP}$ is a language of* propositional logic *with* propositional symbols*:* $a_0, a_1, \ldots$*;* connectives*: "," (*conjunction*) and meta-connective ";";* disjunction*, denoted as "|"; $\leftarrow$ (*derivation*, also denoted as $\rightarrow$);* propositional constants $\bot$ *(*falsum*); $\top$ (*verum*); "$\neg$" (*default negation *or* weak negation*, also denoted with the word* not*); "$\sim$" (*strong negation*, equally denoted as "$-$");* auxiliary symbols*: "(", ")" (*parentheses*). The propositional symbols are called* atoms *too or* atomic propositions. *A* literal *is an atom or a* strong-negated atom. *A* rule *is an ordered pair* $\mathit{Head}(\rho) \leftarrow \mathit{Body}(\rho)$.

An intuitive meaning of *strong negation* "$\sim$" in logic programs with respect to the default negation "$\neg$" is the following: a rule

$$\rho_0 \leftarrow \neg\rho_1$$

allows to derive ρ_0 when there is no *evidence* of ρ_1, while a rule like $\rho_0 \leftarrow \sim\rho_1$ derives ρ_0 only when there is an *evidence* for $\sim\rho_1$, i.e. when it can be proved that ρ_1 is false.

With the notation introduced in Definition 2.1, one may construct clauses of the following general form that are well known in the literature.

Definition 2.2 (Extended Disjunctive Logic Program, EDLP). *An* extended disjunctive logic program *is a set of rules of form*

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \neg\ell_{m+1}, \ldots, \neg\ell_n \tag{1}$$

where ℓ_i is a literal and $0 \leq l \leq m \leq n$.

Naturally, an *extended logic program* (or ELP hereafter) is a finite set of rules of form (1) with $l = 1$; while an *integrity constraint* (also known in the literature as *strong constraint*) is a rule of form (1) with $l = 0$. In particular, for a literal ℓ, the *complementary literal* is $\sim\ell$ and vice versa; for a set $\mathcal{M}$ of literals, $\sim\mathcal{M} = \{\sim\ell \mid \ell \in \mathcal{M}\}$, and $Lit_{\mathcal{M}}$ denotes the set $\mathcal{M} \cup \sim\mathcal{M}$; finally, a *signature* $\mathfrak{L}_\Pi$ is a finite set of literals occurring in Π. Additionally, given a set of literals $\mathcal{M} \subseteq \mathcal{A}$, the complement set $\overline{\mathcal{M}} = \mathcal{A} \setminus \mathcal{M}$.

The well-known *semantics of an EDLP* consists of reducing general rules to rules without default negation "$\neg$" because the latter can be interpreted in *classical logic* by means of the well-known *Herbrand models*. In particular, the reduced rules with no default negation Mon of a rule of the form (1) is

$$\ell_1 \vee p_1 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m \tag{2}$$

where ℓ_i are literals and $0 \leq l \leq m$. This kind of rules is known in the literature as *monotonic counterpart* or *positive program*. Additionally, the monotonic counterpart of a set of rules is the set of the monotonic counterparts of its rules.

Now let us introduce the meaning of programs with both monotonic and nonmonotonic counterparts.

Suppose a finite ground program Π, consisting of clauses of form (1). For any set $\mathcal{S} \subseteq \mathfrak{L}_\Pi$, the *answer-sets reduct* $\Pi^{\mathcal{S}}$ corresponds to

$$\Pi^{\mathcal{S}} = \{\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m \mid \{\ell_{m+1}, \ldots, \ell_n\} \cap \mathcal{S} = \emptyset\} \tag{3}$$

Stating $\mathcal{S}$ as a set of literals rather than atoms, makes one of the differences with *Stable-Models* semantics.

Next, the meaning of a monotonic counterpart corresponds to its *minimal classical model* as follows.

Definition 2.3 (Minimal Closure, Cn(Π)). *Let* Π *be a* positive extended disjunctive program *and* $\mathfrak{L}_\Pi$ *the* signature *(set of all ground literals) from* Π. *The set* Cn(Π) *denotes the minimal subset of* $\mathfrak{L}_\Pi$ *where*

1. *for each ground clause* $p_0 \vee p_1 \vee \cdots \vee p_l \leftarrow q_1, \ldots, q_m$ *in* Π, $q_1, \ldots, q_n \in \mathcal{S}$ *implies* $p_i \in \mathcal{S}$ *for some* $0 \leq i \leq l$*; and for each ground clause of the form*

$$\bot \leftarrow q_1, \ldots, q_m \tag{4}$$

   $\{q_1, \ldots, q_m\} \nsubseteq \mathcal{S}$.

2. *if* $\mathcal{S}$ *contains a pair of complementary literals, then* $\mathcal{S} = \mathfrak{L}_\Pi$.

Note that item (2.) in this Definition 2.3 extends *Stable Models* by giving a meaning to *strong negation*.

Finally, an *answer set* of a given program Π is a *minimal closure* of its reduct as following stated.

Definition 2.4 (Answer Set). *Suppose* Π *is a EDLP and* $\mathcal{S}$ *a set of literals. Then,* $\mathcal{S}$ *is an* answer set *of* Π *if and only if* $\mathcal{S} = Cn(\Pi^{\mathcal{S}})$.

Notice that all stable models can be viewed as *minimal Herbrand models* of a set of first-order sentences, but not the converse. Additionally, $\mathcal{S}$ is a *consistent answer set* of a given program Π if it does not contain a complementary pair of literals.

2.2 Equivalence in Logic Programming

Checking equivalence between logic programs is of great value, especially when it comes to simplifying them by ignoring some portions of code that might be *redundant* and time-consuming. Moreover, there is a close relation between a particular kind of *equivalence and updates of logic programs*, as discussed along this section.

There are several kinds of equivalence in the literature, particularly in ASP and *monotonic logics* [LPV01, ONA01, IS04, EFST05]. Since ASP programs may be expressed in some monotonic logics, one may take advantage of checking equivalence in either system. In this paper I use $\mathcal{N}_2$-logic as one of its fundamental basis that characterises ASP, as well as a translation function between programs and $\mathcal{N}_2$ theories. The set of axioms , as well as the interpretation of the original AGM postulates in that logic are available in [Pea97, Pea99a] and [OC07], respectively.

When establishing a relation between $\mathcal{N}_2$ and ASP, a translation function between ASP programs and $\mathcal{N}_2$ theories is necessary. The function is similar to the one from [LPV01]:

Definition 2.5 (Translation into Nelson's logic [LPV01]). *The mapping function* $T_{\mathcal{N}_2}(\cdot)$ *translates an EDLP into propositional formulas of Nelson's logic* $\mathcal{N}_2$.

The rule

$$p_0 \vee p_1 \vee \cdots \vee p_l \leftarrow q_1, \ldots, q_m, \neg q_{m+1}, \ldots, \neg q_n$$

is mapped into the formula

$$(q_1 \wedge \cdots \wedge q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n) \supset p_0 \vee p_1 \vee \cdots \vee p_l$$

and the strong-negation *propositional constant "$\sim$" has the same meaning of the logical constant "$\sim$" in* $\mathcal{N}_2$.

With this translation, one may redefine ASP in terms of $\mathcal{N}_2$-logic, which shall be useful to provide even more features, discussed along this section.

Let us briefly give a general picture on how the results of such a characterisation arose: From the logics side, [Nel49] defined $\mathcal{N}_2$ by introducing *constructive falsity* into *Intuitionistic logic*, $\mathcal{H}_{\mathsf{I}}$, and defined $\mathcal{N}_2$ as a least constructive extension to *Here-and-There logic*, $\mathcal{H}_{\mathsf{HT}}$, which can be defined by extending $\mathcal{H}_{\mathsf{I}}$. [Pea99a] calls it *conservative extension* of $\mathcal{H}_{\mathsf{I}}$.

From the ASP side, [Pea99b] discovered a new way to define *stable models* in terms of both *Intuitionistic logic* and in the logic of *Here-and-There* in [Pea99a]. Accordingly, as ASP without strong negation is SM, the former can have the same characterisations of SM and vice versa[3]. Besides these characterisations, [Pea99a] discovered that by introducing *strong negation*, like in Definition 2.3, Nelson's $\mathcal{N}_2$-logic characterises ASP as well.

The main result of such characterisations that is more relevant to this paper, can be expressed by Theorem 2.1, by using the notation from [OC07].

To begin with, the notation $\tau \Vdash_{\mathcal{N}_2} \mathcal{M}$ is a shorthand for both τ is *consistent* and *derives* $\mathcal{M}$ in $\mathcal{N}_2$-logic.

Theorem 2.1 ([OC07]). *Let Π be a program over a set of atoms $\mathcal{A}$ and $\mathcal{M} \subseteq \mathfrak{L}_{\mathcal{A}}$ a consistent set of literals. The set $\mathcal{M}$ is an answer set of Π if and only if $T_{\mathcal{N}_2}(\Pi) \cup \neg\overline{\mathcal{M}} \cup \neg\neg\mathcal{M} \Vdash_{\mathcal{N}_2} \mathcal{M}$.*

With this theorem, one may easily establish an equivalence relation between ASP programs for some upcoming update properties:

Theorem 2.2 ([LPV01]). *For any programs Π_1 and Π_2, $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ if and only if for every program Π, $\Pi \cup \Pi_1$ and $\Pi \cup \Pi_2$ have the same Answer Sets.*

In order to simplify notation and with a slight abuse of notation, for any ASP programs Π_0, Π_1, $\Pi_0 \equiv_{\mathcal{N}_2} \Pi_1$ shall actually stand for $T_{\mathcal{N}_2}(\Pi_0) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_1)$.

2.3 Weak Constraints

[LPF+06] introduced a nice feature of DLV solver known as *Weak Constraints* that may be employed to set up preferences between models. In particular, a *weak constraint* is a variant of an *integrity constraint* that may be violated in order to establish priorities amongst models. One of its differences is the introduction of a new derivation symbol "$:\sim$", rather than "$:-$ " or "$\leftarrow$". Moreover, one can specify the priority level and weight of the constraint. Formally,

Definition 2.6 (Weak Constraint [LPF+06]). *A* weak constraint *(ω) is an expression of the form*

$$:\sim \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m[w : p] \tag{5}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables.

In addition, $\Omega(\Pi)$ shall denote the finite *set of weak constraints* occurring in a given program Π. Likewise, a *ω-program* is a logic program with weak constraints.

[3] Note that *strong negation* can be introduced in SM very easily, with extra atoms and *integrity constraints*.

In order to provide a more syntactic sugar, another way to define a weak-consitraint expression from Definition 2.6 is as follows.

Definition 2.7 (Weak Constraint). *A* weak constraint *(ω) is an expression of the form*

$$[w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m \tag{6}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables.

From now on, the previous weak-constraints form shall be employed in the context of DLV-code, while the other in higher abstraction levels.

Similarly to integrity constraints in Section 2.1, one may say that a weak constraint $\rho = ([w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m)$ is *violated* by an answer set $\mathcal{S}$ of a program Π if the following three conditions hold:

1. $\rho \in \Pi$
2. $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{S}$
3. $\{\ell_{k+1}, \ldots, \ell_m\} \nsubseteq \mathcal{S}$

Additionally, [LPF+06] simplify the combination of weights in levels by introducing a function $H^{\Pi}(\mathcal{S})$ that grows in direct proportion to the weight and level of the weak constraint as follows:

Definition 2.8 (Objective Function, $H^{\Pi}(\mathcal{S})$ [LPF+06]). *Given a ground program Π with weak constraints $\Omega(\Pi)$ and an answer set $\mathcal{S}$, the ω* objective function *$H^{\Pi}(\mathcal{S})$ is defined by using an auxiliary function f_{Π} that maps levelled weights to weights without levels:*

$$\begin{aligned} f_{\Pi}(1) &= 1 \\ f_{\Pi}(n) &= f_{\Pi}(n-1) \cdot |\Omega(\Pi)| \cdot w^{\Pi}_{max} + 1, n > 1 \\ H^{\Pi}(\mathcal{S}) &= \sum_{i=1}^{l^{\Pi}_{max}} (f_{\Pi}(i) \cdot \sum_{\rho \in N^{\Pi}_i(\mathcal{S})} weight(\rho)) \end{aligned}$$

where $N^{\Pi}_i(\mathcal{S})$ denotes the weak constraints at level i violated by $\mathcal{S}$, and $weight(\rho)$ the weight of weak constraint ρ.

Finally, the *best models* of such a logic program are those that *minimise* the number of *violated weak constraints*.

Definition 2.9 (Weak-Constraint Model [LPF+06]). *For an EDLP Π with weak constraints, a set $\mathcal{S}$ is a* weak-constraint model *of Π if and only if*

1. *$\mathcal{S}$ is an answer set of Π*

2. $H^{\Pi}(\mathcal{S})$ *is minimal over all the answer sets of* Π.

When the underlying semantics is ASP in Definition 2.9, a weak-constraint model is also known as *Optimal Answer Set*.

Moreover, the language of EDLP's with weak constraints shall be called DATALOG$^{\vee,\omega}$, which is very similar to the notation from the literature.

2.4 Abductive Programming and GAS

As one of the semantics to interpret abductive programs, *Minimal Generalised Answer Sets* (MGAS) provides a more general and flexible semantics than standard ASP, with a wide range of applications. This framework is briefly introduced in the following set of definitions.

Definition 2.10 (Abductive Logic Program [KM90]). *An* abductive logic program *is a pair* $\langle \Pi, \mathcal{A}^* \rangle$ *where* Π *is an arbitrary program and* $\mathcal{A}^*$ *a set of literals, called abducibles.*

On the other hand, there already exists a semantics to interpret abductive programs, called *generalised answer sets* (GAS) due to [KM90].

Definition 2.11 (GAS [KM90]). *The expression* $\mathcal{M}(\Delta)$ *is a* generalised answer set *of the abductive program* $\langle \Pi, \mathcal{A}^* \rangle$ *if and only if* $\Delta \subseteq \mathcal{A}^*$ *and* $\mathcal{M}(\Delta)$ *is an answer set of* $\Pi \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$.

In case of more than one generalised answer sets, a *preferred inclusion order* may be established.

Definition 2.12 (Abductive Inclusion Order [BG03]). *Let* $\mathcal{M}(\Delta_1)$ *and* $\mathcal{M}(\Delta_2)$ *be generalised answer sets of* $\langle \Pi, \mathcal{A}^* \rangle$. *The relation* $\mathcal{M}(\Delta_1) \leq_{\mathcal{A}^*} \mathcal{M}(\Delta_2)$ *holds if and only if* $\Delta_1 \subseteq \Delta_2$.

Last, one can easily establish the *minimal generalised answer sets* from an abductive inclusion order with the following definition

Definition 2.13 (MGAS [BG03]). *Let* $\mathcal{M}(\Delta)$ *be a* minimal generalised answer set *(MGAS) of* $\langle \Pi, \mathcal{A}^* \rangle$ *if and only if* $\mathcal{M}(\Delta)$ *is a generalised answer set of* $\langle \Pi, \mathcal{A}^* \rangle$ *and it is minimal with respect to its* abductive inclusion order.

This simple and strong framework is the main core of a solid foundation for the update formulation, presented in the following sections.

3 $\otimes$-Operation

Intuitively, this approach consists in relaxing all rules in previous programs to the latest update, with a unique abducible. As a result, a transformed *relaxed program* is part of an abductive program, and the other part is the set of

abducibles. Out of the corresponding GAS's of the abductive program, one may get the minimal with respect to its *sequence order* —MSGAS. Finally, the expected model should expressed in its original alphabet out of the MSGAS's.

An α-*relaxed rule* is a rule ρ that is *weakened* by a default-negated atom α in its body: $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho) \cup \{\neg\alpha\}$. In addition, an α-*relaxed program* is a set of α-relaxed rules.

In the particular case of sequences, there are both relaxed sequences and a relaxed program, as formally expressed below.

The α-*relaxed program* of a sequence of ELP's, $(\Pi_1, \Pi_2, \ldots, \Pi_n)$, over a set of atoms $\mathcal{A}$, is the set $\Pi' = \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1}$, where Π'_i is the α-*relaxed program* of Π_i by a new unique abducible $\alpha \notin \mathcal{A}$ for each rule $\rho \in \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$ with $1 \leq i \leq n-1$, and $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$ is the corresponding α-*relaxed sequence*.

Moreover, it is necessary to set up an abductive program from the α-relaxed program of an update sequence and establish an order to get the desired properties. The problem consists in favouring those models that have the least number of abducibles at the highest level of the sequence.

The functions introduced in Section 2.3 from DLV's *weak constraints* have more general characteristics and are a good candidate to figure out the problem. Its general intuition consists in extending ASP to include constraints that may be violated. Accordingly, the goal of such an extended program is the optimal model(s) that violates the minimal number of weak constraints at a certain *priority level*.

The goal of the main *objective function*, as called by [LPF+06], is to simplify the combination of weights in priority levels.

This function has a strong relation to GAS, as shown in Section 7.2. As a result, it can be a basis to bring about a *cardinality order*, and simplified to the needs of update sequences as follows.

Definition 3.1 (Abductive Sequence Order; MSGAS). *Given an update sequence of ELP's,* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*, over a set of atoms* $\mathcal{A}$ *with* n *as an integer; with its corresponding* α*-relaxed sequence* $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$*, where* $\alpha \in \mathcal{A}^*$ *and the* α*-relaxed program* Π'*; and the abductive program* $\Pi_{\mathcal{A}^*} = \langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$*:*

- $w(l) = \begin{cases} 1, & l = 1 \\ w(l-1) \cdot |\mathcal{A}^*| + 1 & l > 1 \end{cases}$ *where* l *is a positive integer.*
- $s(\Delta) = \sum_{l=1}^{n-1} (w(l) \cdot |\{\alpha \mid \alpha \in \mathfrak{L}_{\Pi'_l}\}|)$*, where* $\mathcal{M}(\Delta)$ *is a GAS of* $\Pi_{\mathcal{A}^*}$ *and* $\alpha \in \Delta$.
- $\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) \leq s(\Delta_2)$.
- $\mathcal{M}(\Delta_1) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) = s(\Delta_2)$.
- $\mathcal{M}(\Delta)$ *is a* Minimal Sequenced Generalised Answer Set*, MSGAS, of* $\Pi_{\mathcal{A}^*}$ *if and only if* $\mathcal{M}(\Delta)$ *is minimal with respect to* $\leq_{\mathcal{S}}$.

Let us illustrate this definition with the following simple sequence of updates:

Example 3.1. *Suppose the sequence* $\Pi_1 \otimes \Pi_2 \otimes \Pi_3$ *where*

$$\begin{aligned}\Pi_1 = &\quad \{(a \leftarrow \top), (b \leftarrow \top)\}\\ \Pi_2 = &\quad \{\sim b \leftarrow \top\}\\ \Pi_3 = &\quad \{b \leftarrow b, \neg c\}\end{aligned}$$

Its α*-relaxed sequence*

$$\begin{aligned}\Pi_1' = &\quad \{(a \leftarrow \neg\alpha_1), (b \leftarrow \neg\alpha_2)\}\\ \Pi_2' = &\quad \{\sim b \leftarrow \neg\alpha_3\}\end{aligned}$$

and its α*-relaxed program*

$$\begin{aligned}\Pi' = &\quad \{(a \leftarrow \neg\alpha_1), (b \leftarrow \neg\alpha_2),\\ &\qquad \sim b \leftarrow \neg\alpha_3\}\end{aligned}$$

The abductive program $\langle\, \Pi' \cup \Pi_3, \mathcal{A}^* \,\rangle$ *has the following GAS's:*

$$\begin{aligned}\mathcal{M}(\Delta_1) = \{a, \sim b\}_{\{\alpha_2\}}; &\qquad \mathcal{M}(\Delta_2) = \{\sim b\}_{\{\alpha_1,\alpha_2\}};\\ \mathcal{M}(\Delta_3) = \{a, b\}_{\{\alpha_3\}}; &\qquad \mathcal{M}(\Delta_4) = \{b\}_{\{\alpha_1,\alpha_3\}};\\ \mathcal{M}(\Delta_5) = \{a\}_{\{\alpha_2,\alpha_3\}}; &\qquad \mathcal{M}(\Delta_6) = \{\}_{\{\alpha_1,\alpha_2,\alpha_3\}}\end{aligned}$$

where

$$\begin{aligned}\Delta_1 = \{\alpha_2\}; &\qquad \Delta_2 = \{\alpha_1, \alpha_2\};\\ \Delta_3 = \{\alpha_3\}; &\qquad \Delta_4 = \{\alpha_1, \alpha_3\};\\ \Delta_5 = \{\alpha_2, \alpha_3\}; &\qquad \Delta_6 = \{\alpha_1, \alpha_2, \alpha_3\}\end{aligned}$$

and each w *has the following weights:*

$$w(1) = 1; w(2) = 4$$

and the corresponding weights for each Δ *are*

$$\begin{aligned}s(\Delta_1) = 1; &\qquad s(\Delta_2) = 2;\\ s(\Delta_3) = 4; &\qquad s(\Delta_4) = 5;\\ s(\Delta_5) = 5; &\qquad s(\Delta_6) = 6\end{aligned}$$

According to the sequence, the order of these GAS's is

$$\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2) \leq_{\mathcal{S}} \mathcal{M}(\Delta_3) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4) \leq_{\mathcal{S}} \mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_6)$$

Note that $\mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4)$ *holds as well! Then,* $\mathcal{M}(\Delta_4) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_5)$.
Finally, $\{a, \sim b\}_{\{\alpha_2\}}$ *is its unique MSGAS.*

Strictly speaking, $\leq_{\mathcal{S}}$ is a *pre-order relation* (also known as *preorder* and *quasiorder*) because it is reflexive and transitive. However, $\leq_{\mathcal{S}}$ is mapped into a total-order relation of natural numbers that represent the weight of a preferred model —$s(\Delta)$— as shown in Definition 3.1.

Intuitively, these orders are with respect to the latest update, postulates (R1) and (U1) in [KM89, KM91b, KM91a] and to a *minimal change with respect to cardinality*: MSGAS.

Proposition 3.1. *Abductive Sequence Order ($\leq_{\mathcal{S}}$) is not an antisymmetric relation.*

Proof. Consider the update sequence $\Pi_1 \otimes \Pi_2 \otimes \Pi_3$ from Example 3.1, where two of its GAS's are $\mathcal{M}(\Delta_4) = \{b\}_{\{\alpha_1,\alpha_3\}}$ and $\mathcal{M}(\Delta_5) = \{a\}_{\{\alpha_2,\alpha_3\}}$. Clearly, $\mathcal{M}(\Delta_4) \leq_{\mathcal{S}} \mathcal{M}(\Delta_5)$ and $\mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4)$. However, $\mathcal{M}(\Delta_4) \neq \mathcal{M}(\Delta_5)$. □

From Proposition 3.1 one can easily verify the following result.

Corollary 3.1. *Abductive Sequence Order ($\leq_{\mathcal{S}}$) is a total pre-order relation.*

These are the necessary definitions to formulate an updating sequence of logic programs, proposed in this paper. In short, the formulation consists in the transformation of the update sequence into a single abductive program for which there is a set of *preferred abducibles* according to its order in the relaxed sequence. Finally, *Update Answer Sets* are the models one expects from an updating sequence:

Definition 3.2 ($\otimes$-update Answer Set). *Given a $\otimes$-sequence of updating ELP's $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, with $n \geq 2$, over a set of atoms $\mathcal{A}$, the set $\mathcal{S} \subseteq \mathcal{A}$ is its* $\otimes$-update answer set *if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some* minimal sequenced generalised answer set *$\mathcal{S}'$ of the sequence, and $\otimes$ is the corresponding* update operator.

Let us illustrate the entire framework with a couple of more examples.

Example 3.2. *Suppose an update to Π_1 with Π_2, with Π_3 with Π_4 and with Π_5, where $\Pi_1 = \{b \leftarrow \top\}; \Pi_2 = \{\sim b \leftarrow \top\}; \Pi_3 = \{b \leftarrow b, \neg c\}; \Pi_4 = \{a \leftarrow \top\}; \Pi_5 = \{\sim a \leftarrow \top\}$, from which one would expect $\{\sim a, \sim b\}$ as its **unique update answer set**. The abductive program of the sequence corresponds to $\langle\, \Pi' \cup \Pi_5, \mathcal{A}^* \,\rangle$, where $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and*

$$\begin{aligned}
\Pi' = \{ b &\leftarrow \neg\alpha_1 && (7)\\
\sim b &\leftarrow \neg\alpha_2 && (8)\\
b &\leftarrow b, \neg c, \neg\alpha_3 && (9)\\
a &\leftarrow \neg\alpha_4 \} && (10)
\end{aligned}$$

whose GAS's are

$$\mathcal{M}(\Delta_1) = \{\sim a, \sim b\}_{\{\alpha_1,\alpha_4\}}; \qquad \mathcal{M}(\Delta_2) = \{\sim a, \sim b\}_{\{\alpha_1,\alpha_3,\alpha_4\}};$$
$$\mathcal{M}(\Delta_3) = \{\sim a, b\}_{\{\alpha_2,\alpha_4\}}; \qquad \mathcal{M}(\Delta_4) = \{\sim a, b\}_{\{\alpha_2,\alpha_3,\alpha_4\}};$$
$$\mathcal{M}(\Delta_5) = \{\sim a\}_{\{\alpha_1,\alpha_2,\alpha_4\}}; \qquad \mathcal{M}(\Delta_6) = \{\sim a\}_{\{\alpha_1,\alpha_2,\alpha_3,\alpha_4\}}$$

and

$$\Delta_1 = \{\alpha_1, \alpha_4\}; \qquad \Delta_2 = \{\alpha_1, \alpha_3, \alpha_4\};$$
$$\Delta_3 = \{\alpha_2, \alpha_4\}; \qquad \Delta_4 = \{\alpha_2, \alpha_3, \alpha_4\};$$
$$\Delta_5 = \{\alpha_1, \alpha_2, \alpha_4\}; \qquad \Delta_6 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$$

where each w has the following weights:

$$w(1) = 1; w(2) = 5; w(3) = 21; w(4) = 85$$

and the corresponding weights for each Δ are

$$s(\Delta_1) = 86; \qquad s(\Delta_2) = 107;$$
$$s(\Delta_3) = 90; \qquad s(\Delta_4) = 111;$$
$$s(\Delta_5) = 91; \qquad s(\Delta_6) = 112$$

*Next, its unique MSGAS clearly is $\mathcal{M}(\Delta_1)$, and last, its **update answer set** is just $\{\sim a, \sim b\}$, as one would expect.*

Finally, the problem introduced in [ABBL05]recapitulated in Example 1.1 and solved in [GDOZ05, Gua08], can easily be modelled with this current sequencial approach:

Example 3.3. *Consider the update sequence $\Pi_1 \otimes \Pi_2$ from Example 1.1, whose abductive program may be coded into $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$.*

$$\begin{aligned} \Pi' = \{ day &\leftarrow \neg night, \neg\alpha_1 \\ night &\leftarrow \neg day, \neg\alpha_2 \\ stars &\leftarrow night,\ \neg cloudy, \neg\alpha_3 \\ \sim stars &\leftarrow \neg\alpha_4 \} \end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$.

When the abductive program is interpreted, there are 16 possible combinations ($2^{|\mathcal{A}^*|}$) to include the abducibles with the following GAS's in this case:

$$\{\}_{\{\alpha_1,\alpha_2,\alpha_3,\alpha_4\}}, \{\sim stars\}_{\{\alpha_1,\alpha_2,\alpha_3\}}, \ldots, \{day\}_{\{\alpha_4\}}, \{\sim stars, day\}_{\{\}}$$

Next, the corresponding weights for each w are

$$w(1) = 1; w(2) = 5; w(3) = 21; w(4) = 85$$

*As a result, from the complete list of GAS, it is easy to realise that its unique MSGAS is $\{\sim stars, day\}$, and its **update answer set** $\mathcal{S} = \{\sim stars, day\}$, as one would expect.*

This section consists of the main definitions of an *update semantics for sequences of programs* and they have been illustrated with a series of examples. The following natural step to understand this approach is by means of its main properties that characterise it as a good candidate to represent *dynamic knowledge*.

4 ⊗-Properties

This section shows the main results of the semantics for update sequences that satisfies basic structural properties introduced in [EFST00, EFST02], as well as postulates of *Weak Irrelevance of Syntax* and *Strong Consistency*, just as ⊘-operator in [GDOZ05]. Notwithstanding, this current ⊗-operator can already deal with *multiple updates in a sequence*, as shown in Theorem 4.2 below.

As mentioned before, this set of properties can overcome the sort of problems introduced with Example 1.1 of having extra models when updating with either *redundant information* or *tautological rules*. As a consequence, any semantics of logic programs aimed to be generally accepted should meet at least the basic set of properties introduced in [GDOZ05, GDO06] and summarised below.

Accordingly, one can formulate the following theorem with the properties interpreted for and extended to sequences of updates.

4.1 Inconsistencies

Before proving theorem 4.2 for the called structural properties, it is necessary to introduce some preliminary results of this work that have to do with *inconsistencies*. Firstly, any consistent update to a knowledge base guaranties a consistent result. Formally,

Lemma 4.1 (⊗-weak consistency view)**.** *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's and an update sequence* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *with its corresponding abductive program* $\Pi_{\mathcal{A}^*} = \langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$*. If* Π_n *is consistent then* $\Pi_{\mathcal{A}^*}$ *is consistent.*

sketch. Suppose $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are ELP's and Π_n consistent. This means that $\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$ has a generalised answer set $\mathcal{M}(\Delta)$ out of answer sets of $\Pi' \cup \Pi_n \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$, which is clearly consistent. Therefore, if Π_n is consistent, $\Pi_{\mathcal{A}^*}$ is also consistent. □

Corollary 4.1 (⊗-consistency Preservation)**.** *Suppose that* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's. The update sequence* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *is consistent if* Π_n *is consistent.*

Corollary 4.1 also proves to be useful to *restore consistency* from an originally *inconsistent knowledge base*. This property is also known in the literature as *consistency preservation* and is a general case of [SI03]'s *inconsistency*

removal. Note that the latter's sole name confirms the *syntactical orientation* of their approach. Moreover, the latter requires that $\Pi_0 \otimes \Pi_1 \subseteq \Pi_0$, which isn't applicable to $\otimes$-operator.

As a result, the following proposition follows directly from Corollary 4.1.

Proposition 4.1 (consistency restoration). *Suppose* Π_0 *is an ELP's. The update* $\Pi_0 \otimes \emptyset$ *is consistent.*

Finally, the only reason to have an *inconsistency* in an $\otimes$-update sequence is by having an *inconsistent update*:

Proposition 4.2. *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's. If* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *has no update answer sets, then* Π_n *has no answer sets.*

This preliminary results make a solid framework to introduce the main theorem of this section: meeting the set of structural properties studied along this research.

4.2 Structural Properties

Before listing these properties, it is necessary to recall from Section 2.2 that the statement $\Pi_1 \equiv \Pi_2$ that means that both Π_1 and Π_2 have the same answer sets —or alternatively $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$. With an abuse of notation, when stating equivalence between update sequences, indeed it means that they have the same (or different) *Update Answer Sets*. Last but not least, notice that equivalence between update sequences of more than two programs seems to make sense at the level of weak equivalence rather than strong, and that is because each update sequence means one and only one update. As a consequence, there is no resulting knowledge base that might contain incomplete information.

On the other hand, another useful theorem from [LPV01] is also necessary:

Theorem 4.1 ([LPV01]). *For any programs* Π_1 *and* Π_2, $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ *if and only if for every program* Π, $\Pi \cup \Pi_1$ *and* $\Pi \cup \Pi_2$ *have the same Answer Sets.*

In order to simplify notation and with a slight abuse of notation, for any ASP programs Π_0, Π_1, $\Pi_0 \equiv_{\mathcal{N}_2} \Pi_1$ shall actually stand for $T_{\mathcal{N}_2}(\Pi_0) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_1)$.

$\otimes$-SP-1, Addition of Tautologies [EFST02]: If Π has only *tautological clauses* of the form $\ell \leftarrow \ell$, any *consistent sequence*

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

$\otimes$-SP-2, Initialisation [EFST02]:

$$\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-**SP-3, Inertia:** If Π_n is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-**SP-4, Idempotence:** For any sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-**SP-6, Non-interference:** If $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over mutually *disjoint alphabets*, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$$

⊗-**SP-7, Augmented Update [EFST02]:** If $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$$

⊗-**SP-8, Strong Consistency, SC :** If $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ with $n \geq 2$ is consistent, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$$

⊗-**SP-8a, Weak Consistency, WC :** If $\Pi_n \cup \Pi_m$ is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$$

This is a particular case of property ⊗-SP-8, applicable to sequences of updates with $n > 2$.

⊗-**SP-9, Weak Irrelevance of Syntax, WIS:** If $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$$

This is the set of minimal structured properties that an update semantics should meet in order to avoid counterintuitive results like the ones in Example 3.3. With this collection one can formulate the following theorem.

Theorem 4.2. *Given a ⊗-sequence of update extended logic programs $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, with $n \geq 2$, over a set of atoms $\mathcal{A}$, ⊗-operator satisfies properties ⊗-SP-1 to ⊗-SP-9.*

Proof for Theorem 4.2. ⊗-**SP-2, Initialisation:** $\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$.

$\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the abductive program $\langle \emptyset \cup \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi_n, \mathcal{A}^* \rangle$, that is the same abductive program $\langle \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi_n, \mathcal{A}^* \rangle$ from $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ and both of them have the same GAS's, and the same MSGAS's and thus the same ⊗-update answer sets. Hence, $\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ as required.

$\otimes$-SP-3, Inertia: $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$.

Assume Π_n consistent. Then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \emptyset, \mathcal{A}^* \,\rangle$, which is logically equivalent to the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. This means that $\mathcal{M}(\Delta) \leq_{\mathcal{S}} \mathcal{M}(\Delta')$ and that $\mathcal{M}(\Delta)$ is an answer set of $\Pi'_1 \cup \Pi'_2 \cup \ldots \Pi'_n \cup \{\alpha \leftarrow \top\}$ where $\alpha \in \Delta$ and $\Delta \cap \mathfrak{L}_{\Pi'_n} = \emptyset$. Thus, $s(\Delta) \leq s(\Delta')$ and $w(x) \leq w(n)$ for $x < n$, which is clearly true. Then $\mathcal{M}(\Delta)$ is also a MSGAS from the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$ out of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. That means both update sequences have the same $\otimes$-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ as required.

$\otimes$-SP-8a, Weak Consistency: If $\Pi_n \cup \Pi_m$ is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$$

Suppose $\Pi_n \cup \Pi_m$ is consistent. Then, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \Pi_m, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. Since $\Pi_n \cup \Pi_m$ is consistent, $\mathcal{M}(\Delta) \leq_{\mathcal{S}} \mathcal{M}(\Delta')$ with $\Delta \cap \mathfrak{L}_{\Pi'_n} = \emptyset$, and $s(\Delta) \leq s(\Delta'); w(x) \leq w(n); x < n$. Thus, $\mathcal{M}(\Delta)$ is the same MSGAS of the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n \cup \Pi_m, \mathcal{A}^* \,\rangle$ out of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m$. This means that both update sequences have the same $\otimes$-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$.

$\otimes$-SP-8, Strong Consistency: If $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ with $n \geq 2$ is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$.

Assume $\mathcal{M}$ is an answer set of $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ and $n \geq 2$. Then, $\mathcal{M}$ must be the same model of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. As $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the abductive program $\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$, and an arbitrary answer set of $\Pi' \cup \Pi_n \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ is $\mathcal{M}(\Delta)$, then the MSGAS's of $\langle \Pi' \cup \Pi_n, \mathcal{A}^* \rangle$ are just $\mathcal{M}(\emptyset)$. Because the MSGAS is $\mathcal{M}(\emptyset)$, then the literals in $\mathfrak{L}_{\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle} \cap \mathcal{A}^*$ are not positive and Π' is an ordinary extended logic program whose semantics coincides with $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$, as required.

$\otimes$-SP-1, Addition of Tautologies [EFST02]: If Π has only *tautological rules* of the form $\ell \leftarrow \ell$, any *consistent sequence*

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

Suppose $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ consistent. This means that $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \Pi, \mathcal{A}^* \,\rangle$ that is logically equivalent to $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. Then by $\otimes$-SP-3, $\mathcal{M}(\Delta)$ are also the same MSGAS's than the ones from

the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$, which is the abductive program of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. This means both update sequences have the same $\otimes$-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, as required.

$\otimes$-SP-4, Idempotence: For any sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

Suppose $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has update answer sets. Then, the update sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the $\otimes$-abductive program $\Pi_{\mathcal{A}^*} = \langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi'_n \cup \Pi''_1 \cup \Pi''_2 \cup \cdots \cup \Pi''_{n-1} \cup \Pi_n, \mathcal{A}^* \,\rangle$ where Π''_z is the relaxed program of Π_z such that $(\Pi''_z \cap \mathcal{A}^*) \cap (\Pi'_z \cap \mathcal{A}^*) = \emptyset$ and thus, the weakened rules of Π_z by Π'_z are the same weakened rules by Π''_z in every GAS of $\Pi_{\mathcal{A}^*}$. Hence, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, as required. Therefore, for a sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

$\otimes$-SP-9, Weak Irrelevance of Syntax: If $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y.$$

Suppose $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ Then, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x$ and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$ have the respective abductive programs $\langle\, \Pi' \cup \Pi_x, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_y, \mathcal{A}^* \,\rangle$, whose respective GAS's come from programs $\Pi' \cup \Pi_x \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ and $\Pi' \cup \Pi_y \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ with $\Delta \subseteq \mathcal{A}^*$ and with the same set of abducibles. By Theorem 4.1, $\Pi' \cup \Pi_x \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\} \equiv \Pi' \cup \Pi_y \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$. Consequently, $\langle\, \Pi' \cup \Pi_x, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_y, \mathcal{A}^* \,\rangle$ have the same GAS's, the same MSGAS's, and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$, as required. Therefore, if $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

$\otimes$-SP-7, Augmented Update: If $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

Suppose Π_y is consistent and $\Pi_x \subseteq \Pi_y$. This implies that $\Pi_x \cup \Pi_y = \Pi_y$ and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$ is logically equivalent to $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \cup \Pi_y$. Then, by weak consistency, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$, as required. Therefore, if $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

$\otimes$-SP-6, Non-interference: If $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over mutually*disjoint alphabets*, and either all of them are consistent or are not, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$$

Assume that $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are defined over disjoint alphabets and that all of them are consistent. Then, both $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ and $\Pi_n \cup \Pi_{n-1} \cup \cdots \cup \Pi_1$ are consistent too and by the commutative law on the union[4] , they may have any order. Thus, by Strong Consistency on any of the orders of the union, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n \equiv \Pi_n \cup \Pi_{n-1} \cup \cdots \cup \Pi_1 \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$, as required.

Now suppose that $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ all are inconsistent. Then, the $\otimes$-update sequences $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ and $\Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$ have their respective abductive programs $\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_1, \mathcal{A}^* \,\rangle$, which are clearly inconsistent and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$, as required.

Therefore, if $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over disjoint alphabets, and either all of them are consistent or not, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$.

□

This is the *minimal set of properties* considered as fundamental for a proper update semantics from [GDOZ05] to [Gua07b], most of them first introduced in [EFST02], and $\otimes$-operator satisfies at least this collection of properties.

One immediate result from Theorem 4.2 is the equivalence between an abductive program of the form $\Pi_{\mathcal{A}^*}$ in Definition 3.1 where Π_n is also α-relaxed and another where Π_n is not relaxed. This is formally expressed in Proposition 4.3 as follows.

Proposition 4.3 (Full Relaxation). *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's and* Π_n *consistent. The abductive program* $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$*, out of the* α*-relaxed programs from the sequence has the same MSGAS's as the abductive program* $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$ *does.*

Another nice property has to do with *updates to inconsistent programs* that shall be formalised in upcoming sections. Meanwhile, consider the following update that recovers consistency of a sequence with an inconsistency.

Example 4.1. *Suppose an update to* Π_1 *with* Π_2*, and with* Π_3*, where* $\Pi_1 = \{a \leftarrow \top\}$, $\Pi_2 = \{(b \leftarrow \top), (\sim b \leftarrow \top)\}$ *and* $\Pi_3 = \{\}$. *Its abductive program corresponds to* $\langle\, \Pi', \mathcal{A}^* \,\rangle$*, where*

$$\begin{aligned} \Pi' = \{a &\leftarrow \neg\alpha_1 \\ b &\leftarrow \neg\alpha_2 \\ \sim b &\leftarrow \neg\alpha_3\} \end{aligned}$$

[4] Notice that associativity law does not apply to $\otimes$-operator.

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3\}$*, whose GAS's are*

$$\begin{aligned} \mathcal{M}(\Delta_1) &= \{a, \sim b\}_{\{\alpha_2\}}; & \mathcal{M}(\Delta_2) &= \{\sim b\}_{\{\alpha_1,\alpha_2\}}; \\ \mathcal{M}(\Delta_3) &= \{a, b\}_{\{\alpha_3\}}; & \mathcal{M}(\Delta_4) &= \{b\}_{\{\alpha_1,\alpha_3\}}; \\ \mathcal{M}(\Delta_5) &= \{a\}_{\{\alpha_2,\alpha_3\}}; & \mathcal{M}(\Delta_6) &= \{\}_{\{\alpha_1,\alpha_2,\alpha_3\}} \end{aligned}$$

and the corresponding weights for each w *are*

$$w(1) = 1; w(2) = 4.$$

Next, the corresponding sum of weights for each Δ *are*

$$\begin{aligned} s(\Delta_1) &= 4; & s(\Delta_2) &= 5; \\ s(\Delta_3) &= 4; & s(\Delta_4) &= 5; \\ s(\Delta_5) &= 8; & s(\Delta_6) &= 9 \end{aligned}$$

that lead to two MSGAS's, $\{a, \sim b\}_{\{\alpha_2\}}, \{a, b\}_{\{\alpha_3\}}$ *from which one can conclude that the update answer sets are just* $\{a, b\}, \{a, \sim b\}$*.*

This example shows how to prevent a knowledge base from *collapse due to an inconsistency*. One solution, as Example 4.1 shows, is introducing a *disjunction* until new evidence supports either conclusion. Another solution, as in the following section, is *relaxing the updating program* as well, which produces the same effect as in the example above.

5 $\otimes'$-Operation

One may find more interesting results when analysing Proposition 4.3 and defining a slight different construction to update sequences of EDLP's.

Definition 5.1. *Given an update sequence of ELP's,* $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$*, over a set of atoms* $\mathcal{A}$ *with* n *as a natural number; with its corresponding* α*-relaxed sequence* $(\Pi'_1, \Pi'_2, \ldots, \Pi'_n)$*, where* $\alpha \in \mathcal{A}^*$*; and the abductive program* $\Pi_{\mathcal{A}^*} = \langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ *with* $\mathcal{M}(\Delta)$ *as the GAS's of* $\Pi_{\mathcal{A}^*}$*; and* $w(l); l; s(\Delta)$ *defined as in Definition 3.1:*

- $\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) \leq s(\Delta_2)$*.*
- $\mathcal{M}(\Delta_1) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) = s(\Delta_2)$*.*
- $\mathcal{M}(\Delta)$ *is an MSGAS of* $\Pi_{\mathcal{A}^*}$ *if and only if* $\mathcal{M}(\Delta)$ *is minimal with respect to* $\leq_{\mathcal{S}}$*.*

Finally, *Update Answer Sets* are the models one expects from an updating sequence:

Definition 5.2 ($\otimes'$-update Answer Set). *Given a $\otimes'$-sequence of updating ELP's, $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$, with $n \geq 2$, over a set of atoms $\mathcal{A}$, the set $\mathcal{S} \subseteq \mathcal{A}$ is its* update answer set *if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some* minimal sequenced generalised answer set *$\mathcal{S}'$ of the sequence, and $\otimes'$ is the corresponding* update operator.

Besides the properties shared with $\otimes$-operator, the following section includes an interesting property of *consistency view* for $\otimes'$-operator.

6 $\otimes'$-Properties

Having introduced Proposition 4.3 and $\otimes'$-operator, the following results illustrate another alternative to *preserving a knowledge base from collapse*, due to an *inconsistency update*.

Proposition 6.1 (Full Relaxation Consistency). *Suppose $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are ELP's and Π'_x as the corresponding α-relaxed program with $1 \leq x \leq n$. The abductive program $\langle \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \rangle$ is consistent.*

This proposition also yields the following obvious result:

Corollary 6.1 (Strong Consistency View). *Suppose $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are ELP's. The update sequence $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$ is consistent.*

Another interesting result is its equivalence with the operator just introduced in Section 3, which shows common properties.

Proposition 6.2. *Given the sequence $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ of EDLP's with Π_n consistent, $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*

Proof. This is a straightforward consequence of Proposition 4.3 and Definition 5.1. □

Corollary 6.2 ($\otimes'$-structural properties). *$\otimes'$ operator satisfies properties $\otimes$-SP-1–$\otimes$-SP-9.*

This section has introduced a more general update operator that, besides satisfying the same properties than $\otimes$-operation, it is also more *robust* when new information that is inconsistent arises.

Notice that Lemma 4.1, together with the resulting corollary and Proposition 4.2 are properties that suggest a classification of this approach as a framework for *belief change* rather than *belief updates*, as suggested in [KM91a]. One of the main goals of this work, however, is *representation of knowledge* in general by producing a framework with an *intuitive behaviour*, even though the title of this paper includes the word "updates" for historical and practical reasons. Another difference from [KM91a] is whether the knowledge base

is evolving in a *changing environment* or not, which would come to a subjective statement. As a result, no strict distinction shall be made in this semantics with respect to *belief revision* and *belief update*. It is also worth noticing that [SI03] have a similar position with respect to such a difference in update semantics, where they suggest ambiguous results may arise when trying to state a clear difference under some contexts [SI03].

Finally, the following sections present an implemented prototype to automatically update sequences of logic programs in ASP for the semantics in Section 3. This sort of programs for ASP problem solving are known as *solvers* in the literature.

7 ⊗-prototype

One of the latest ideas introduced in this paper is choosing between generalised answer sets to overcome disadvantages of previous approaches described in [Gua07c]. However, a solver to automatically compute this semantics is necessary for different demanding proposes that go from a classroom tool and assistant, to implementations of more complex prototypes aimed at exploiting knowledge-management fields and further properties discovery.

7.1 Implementing Updates on `DLV`

There are two major efficient *solvers* to compute ASP with a vast background of implementation and research. They are `DLV` [LPF+06] and `SMODELS` [NS97], and the system proposed in this paper employs the former at a higher abstraction level in order to *update ELP programs*. Towards this end, this section is an introduction to a transformation that may be interpreted in either system with some slight syntactic adaptations[5].

To begin with, an approach to implement an update semantics in MGAS was first introduced in [GDOZ05] by means of *preferred disjunctive logic programs* in [Bre02]'s ODLP and has a *solver for pairs of programs* at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`. However, the mentioned update semantics and thus the final system itself, are limited to *single updates*.

Indeed, a justification from [Gua08]to use ODLP is that there is a solver available named `PSmodels`[6] that is an extension to `SMODELS` [NS97] to compute *preferred answer sets*. Unfortunately, up to the printout of this paper, there is no reliable version of `PSmodels` and the latest one (v.2.26a) endures some few *bugs* under certain circumstances[7]. In addition to the *running solver*,

[5] Refere, for instance, to [LPF+06] for an *equivalence of weak constraints* in `SMODELS`.
[6] `http://www.tcs.hut.fi/Software/smodels/priority/`
[7] Try to compute the preferred models of a simple program like $\{a.\}$.

it is believed that DLV significantly outperforms SMODELS [LPF+06], not to mention that ODLP is such a colossal semantics that can do much more complex tasks than just computing MGAS's, which might compromise the performance of the desired system and mix up its simple formulation.

As a result this section is an introduction to the use of DLV's *Weak Constraints* for their characteristics of *optimisation in preferences* [LPF+06] that enjoys the above benefits of being in DLV with no more extra throughput added to the system.

This section consists of an implementation of the declarative version for updates sequences in Section 3, which is proposed as an alternative to other *syntax-based semantics* described in [Gua07c]. One of the main contributions from this implementation is to use DLV's *Weak Constraints* to compute the model(s) of any update sequence, besides providing an *online solver for public experiments*.

The structure of this section consists of the basic configuration of the system, a description of the employed technology, a discussion of the major process to compute the intended models, as well as a set of examples to illustrate its use. Before going straight to the system description, a short recapitulation of its background is necessary to understand its foundation on ASP, abductive programming from Section 2.4, and in particular DLV's *weak constraints* from Section 2.3.

7.2 Weak-constraints Characterisation

This section is an alternative to the characterisation in [Gua08], and it consists in transforming updates of ELP's into a ω-program. The answer sets of such a program shall prove to coincide with the GAS's of the abductive program from the update. Finally, the *optimal answer sets* of the ω-program shall coincide with the MSGAS's of the abductive program. So, let us start by introducing some more notation.

A *Simple Weak Constraint* (ω') is an expression of the form

$$[w : p] \leftarrow \ell$$

where ℓ is a literal; and w and p are optional weight and level parameters as in Definition 2.6.

A ω' derives the following result with some new notation in advance: read $\mathfrak{L}_{\Omega(\Pi)}$ as the *signature of the weak constraints* occurring in Π (a finite set of literals), while $\mathfrak{L}_{N_1^\Pi(\mathcal{M})}$ stands for the *signature* of the weak constraints in Π at level 1 violated by $\mathcal{M}$. Something worth recalling is that $N_i^\Pi(\mathcal{M})$ denotes the weak constraints at level i violated by $\mathcal{M}$ in Π.

Proposition 7.1. *Suppose a logic program* Π *with weak constraints* $\Omega(\Pi)$ *of the form of a* simple weak constraint *with* $w = p = 1$*; suppose an answer set* $\mathcal{M}$ *of* Π*; and a set of literals* $\Delta \subseteq \mathcal{M} \cap \mathfrak{L}_{\Omega(\Pi)}$*. Then,* $\mathfrak{L}_{N_1^\Pi(\mathcal{M})} = \Delta$.

It is worth mentioning that the *model(s) of a logic program with weak constraints* are also called answer sets in the literature.

Before introducing the translation function, let us extend the well-known standard *signature* definition with $\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$, which means the finite set of literals occurring both in Π and in $\mathcal{A}^*$.

Definition 7.1 (Abductive-$\mathcal{W}$ Translation). *Let* $\langle\,\Pi,\mathcal{A}^*\,\rangle$ *be an abductive logic program. A* translation into a weak-constraint program $\mathcal{W}(\Pi,\mathcal{A}^*)$ *corresponds to*

$$\Pi\cup\{\alpha'\vee\alpha\leftarrow\top\}\cup\{[1:1]\leftarrow\alpha\} \tag{11}$$

where $\alpha\in\mathcal{A}^*$ *and* $\alpha'\notin\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$.

This translation and Proposition 7.1 yield the following useful results for the *implementation* of GAS-semantics in *weak constraints*.

Lemma 7.1. $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ *is a* generalised answer set of an abductive program $\langle\,\Pi,\mathcal{A}^*\,\rangle$ *if and only if* $\mathcal{M}$ *is an answer set of* $\mathcal{W}(\Pi,\mathcal{A}^*)$.

sketch. Only-if part. Suppose $\mathcal{M}$ is an answer set of $\mathcal{W}(\Pi,\mathcal{A}^*)$. By Definition 7.1, there are two cases for the answer sets of the ω-program. That $\mathcal{M}$ is an answer set of $\Pi\cup\{\alpha'\vee\alpha\leftarrow\top\}\cup\{[1:1]\leftarrow\alpha\}$ with $\alpha\in\mathcal{M}$ and by Definition 2.11, $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ corresponds to the answer set of $\Pi\cup\{\alpha\leftarrow\top\}$ and therefore by Definition 2.11, to the GAS of $\langle\,\Pi,\mathcal{A}^*\,\rangle$. On the other hand, with $\alpha\notin\mathcal{M}$, $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ is just an answer sets of $\Pi\cup\{\}$ and thus the GAS of $\langle\,\Pi,\mathcal{A}^*\,\rangle$ —Definition 2.11. In both cases, $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ is a GAS of $\langle\,\Pi,\mathcal{A}^*\,\rangle$, as required.

If part. Suppose $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ is a GAS of $\langle\,\Pi,\mathcal{A}^*\,\rangle$. By Definition 2.11, there are two cases for Δ. That $\Delta\neq\{\}$ means that $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ is an answer set of $\Pi\cup\{\alpha\leftarrow\top\}$ for some $\alpha\in\Delta$ with $\Delta\subseteq\mathcal{A}^*$, which corresponds to the answer set $\mathcal{M}$ of the translated program (Definition 7.1) $\Pi\cup\{\alpha'\vee\alpha\leftarrow\top\}\cup\{[1:1]\leftarrow\alpha\}$ with $\alpha\in\mathcal{M}$. On the other hand, $\Delta=\{\}$ means that $\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi,\mathcal{A}^*\,\rangle}$ is an answer set of $\Pi\cup\{\}$ that corresponds to the answer set $\mathcal{M}$ of the ω-program $\Pi\cup\{\alpha'\vee\alpha\leftarrow\top\}\cup\{[1:1]\leftarrow\alpha\}$ with $\alpha\notin\mathcal{M}$. In both cases, $\mathcal{M}$ is an answer set of $\mathcal{W}(\Pi,\mathcal{A}^*)$, as required. □

The *equivalence between a GAS of an abductive program and an answer set of a* ω*-program* ought to be easier to read after a simple example:

Example 7.1. *Suppose an update of* Π_1 *with* Π_2 *where* $\Pi_1=\{b\leftarrow\top\};\Pi_2=\{a\leftarrow\top\}$. *Its corresponding abductive program is* $\langle\,\Pi'\cup\Pi_2,\mathcal{A}^*\,\rangle$ *where* $\Pi'=\{b\leftarrow\neg\alpha\}$ *and* $\mathcal{A}^*=\{\alpha\}$. *As a result,* $\mathcal{W}(\Pi'\cup\Pi_2,\mathcal{A}^*)=\Pi'\cup\Pi_2\cup\{\alpha'\vee\alpha\leftarrow\top\}\cup\{[1:1]\leftarrow\alpha\}$ *whose answer set* $\mathcal{M}=\{\alpha',a,b\}$. *On the other hand, the GAS of the abductive program is just* $\{a,b\}_\emptyset=\mathcal{M}\cap\mathfrak{L}_{\langle\,\Pi'\cup\Pi_2,\mathcal{A}^*\,\rangle}$.

Up to now, one can compute GAS's by means of ω-programs. However, this proposal of *updates at the object level* requires that the generalised answer sets

are minimal with respect to the number of abducibles, as in Definition 3.1. Thus, the **main result** of this section is the following theorem that shows the *equivalence between MSGAS's and optimal answer sets*, the **core of the implementation** and one more property for the semantics.

Theorem 7.1 (MSGAS–Optimal Answer Set). *Let $\mathcal{M}$ be a set of literals. The set $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a* minimal sequenced generalised answer set *of the abductive program $\langle \Pi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}$ is an* optimal answer set *of $\mathcal{W}(\Pi, \mathcal{A}^*)$.*

sketch. If part. Suppose $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a minimal sequenced generalised answer set of the abductive program $\langle \Pi, \mathcal{A}^* \rangle$. Then, $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a GAS of $\langle \Pi, \mathcal{A}^* \rangle$ and is minimal with respect to $\leq_{\mathcal{S}}$ and by Lemma 7.1 $\mathcal{M}$ corresponds to the answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$ and therefore, to its weak-constraint model — Definition 2.9— that is optimal with respect to cardinality.

Only-if part. Suppose $\mathcal{M}$ is an optimal answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$. Then, by Definition 2.9, $\mathcal{M}$ is minimal with respect to $H^{\Pi}(\mathcal{M})$ and corresponds to the minimal set of violated weak constraints that are exactly the minimal number of abducibles in the generalised answer set $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ of $\langle \Pi, \mathcal{A}^* \rangle$ by Proposition 7.1. □

7.3 The Parser

Differently from the implementation in [Gua08], which has a parser embedded in its `PHP`[8] code, the new parser presented in this section has been compiled in `C` on the MacOS X™ platform at Darwin™ level, as well as for a Linux alternative at the same location: `http://www2.in.tu-clausthal.de/~guadarrama/updates/seqs.html`. The advantage of having a *UNIX binary module* is the ease to be plugged in to other modules so as to form more complex applications.

On the other hand, MacOS X™/Darwin™ is a BSD branch of UNIX, that has a ported set of *`Lex`* and *`Yacc`* utilities in their GNU versions of *`Flex`* and *`Bison`* —respectively. `Flex` is a short name for *Fast Lexical Analyser* that generates code to scan text through *regular-expressions pattern matching*.

In particular, the following specification shows the main tokens implemented for this update solver.

NAME	[–~]?[[:alnum:]_]+
PnSTART	"{"
PnEND	"}"
GETS	":– "
NOT	"not␣"
RULEEND	"."
CONJUNCT	","

[8] This is a script language quite suitable for small processes of dynamic contents on web pages.

DISJUNCT "|"

Tokens like PnSTART and PnEND split the sequence of programs into individual programs, while the rest of the tokens need no explanation.

As soon as `Flex` decomposes the text of a program sequence, its output is taken on by `Yacc`, which gives a meaning to each correct structure of rules and program sequence. In particular, the `Yacc` process specifies a *grammar for update sequences*. It is also responsible for weakening each rule in all programs of the sequence with a new unique atom and establishes a *cardinality-preference relation* amongst such atoms, according to the sequence the rule is in. Last, this process is responsible of an *error-checking mechanism* that verifies the correctness of a given program sequence according to the BNF grammar below.

```
<sequence>      ::= <sequence> '{' <program> '}'
                  |
<program>       ::= <program> <rule>
<rule>          ::= <head> END
                  | <head> GETS <body> END
<head>          ::= <POST>
                  |
<POST>          ::= <LITERAL>
                  | <POST> DISJUNCT <LITERAL>
<body>          ::= <PRE>
                  |
<PRE>           ::= <LITERAL>
                  | NOT <LITERAL>
                  | <PRE> CONJUNCT <LITERAL>
                  | <PRE> CONJUNCT NOT <LITERAL>
<LITERAL>       ::= NAME
                  | NAME '(' NAME ',' NAME ')'
```

As mentioned before, for each rule that is analysed, the system appends a pair of new rules with *weakening atoms* and a weak constraint, in programs previous to the last one in the sequence, as follows:

$$[1:p] \quad \leftarrow \quad \alpha_i \tag{12}$$

$$\sim\alpha_i \mid \alpha_i \quad \leftarrow \quad \top \tag{13}$$

where i represents the i^{th} abducible α and p the p^{th} program, the latter forming a *weight-level relation* that corresponds to $[1:p]$.

The intuition behind this formulation is computing the MSGAS of the abductive program by violating the least number of weak constraints. In addition, the $[1:p]$ relation represents the sequence order from Definition 3.1. That is to say, the models are those that have the least weight-level relation.

Example 7.2. *Suppose the sequence*

$$\begin{aligned}\Pi_1 :&\quad \{a \leftarrow \neg b\}\\ \Pi_2 :&\quad \{(b \leftarrow \neg a);(c \leftarrow a);(d \leftarrow {\sim}a, b)\}\\ \Pi_3 :&\quad \{e \leftarrow \neg b\}\end{aligned}$$

The corresponding abductive program is $\langle\, \Pi' \cup \Pi_3, \mathcal{A}^* \,\rangle$ *where*

$$\begin{aligned}\Pi' = \{&a \leftarrow \neg b, \neg\alpha_1 && b \leftarrow \neg a, \neg\alpha_2 &&\\ &c \leftarrow a, \neg\alpha_3 && d \leftarrow {\sim}a, b, \neg\alpha_4 && e \leftarrow \neg b\}\end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. *Such an abductive program is transformed into a* ω*-program in* DLV *as*

$$\begin{array}{rll} a \leftarrow \neg b, \neg\alpha_1 & {\sim}\alpha_1 | \alpha_1 \leftarrow \top & [1:1] \leftarrow \alpha_1 \\ b \leftarrow \neg a, \neg\alpha_2 & {\sim}\alpha_2 | \alpha_2 \leftarrow \top & [1:2] \leftarrow \alpha_2 \\ c \leftarrow a, \neg\alpha_3 & {\sim}\alpha_3 | \alpha_3 \leftarrow \top & [1:2] \leftarrow \alpha_3 \\ d \leftarrow {\sim}a, b, \neg\alpha_4 & {\sim}\alpha_4 | \alpha_4 \leftarrow \top & [1:2] \leftarrow \alpha_4 \\ e \leftarrow \neg b, \neg\alpha_5 & {\sim}\alpha_5 | \alpha_5 \leftarrow \top & [1:3] \leftarrow \alpha_5 \end{array}$$

This is the final stage before interpreting the *preferred transformed program* in DLV with weak constraints, as explained below.

7.4 The Top Module

The *top module* consists of a display of the original sequence, its transformation to abductive program, as well as the result of interpreting such an abductive program under MSGAS and the update answer sets of the sequence. All in all is coded into a *UNIX script*, with some simple sub-processes that filter in the needed text from the formatted output in DLV. Last, this main module is also responsible of dealing with the *user interface* in HTML by getting the user's input sequence into a *text pane* of a web page and processing it to display the output within a new web page.

In a black-box system approach, the main module consists of an HTML page with a text panel to capture the user input of a sequence of ELP, with each program enclosed in braces "{}". In the same page, there is another text pane to capture switches for DLV. Once the user pushes the process button, the system processes the input text and yields an output divided into the original input sequence; its corresponding abductive program; the GAS's of the abductive program and the corresponding update answer set(s) of the sequence, when they exist.

7.4.1 The Abductive Program

The *abductive program* is, indeed, coded into a *cardinality-preference relation* of weak constraints. The relation consists of a *weakened program*, where each rule has its corresponding *pair of disjunctive abducibles* (13), and a weak constraint (12).

This simple process constructs a *triple rule* at the *parser stage* by keeping a counter for each abducible and one for each program in the sequence, which is displayed once a rule is recognised: the *relaxed rule*, a *disjunctive rule* and a *leveled weak constraint*.

7.4.2 Computing MSGAS's

Computing MSGAS's is a straightforward process that takes the *abductive preference program* from the previous process as an input and passes it on to DLV. The general intuition behind this *optimisation solver* is computing the ASP reduct (Definition 2.4) of the ELP input program that returns none or more answer sets. Then, it chooses the *best weak-constraint model*(s). As mentioned before, the best answer set(s) are those that violate the least number of weak constraints (they all have the same weight) at the highest level.

From Example 7.2, the system returns the following two MSGAS's:

$$\{b\}_{\{-\alpha_1,-\alpha_2,-\alpha_3,-\alpha_4,-\alpha_5\}}, \{a,c,e\}_{\{-\alpha_1,-\alpha_2,-\alpha_3,-\alpha_4,-\alpha_5\}}$$

7.4.3 The Update Answer Sets

Finally, the last stage is a simple filtering with UNIX processes of the abductive atoms that omits them and gives the desired result. For this example, just $\{b\}, \{a,c,e\}$.

7.5 ⊗-Complexity

Now it is time to say a few words about *complexity of the* ⊗*-operator*. One may divide the main problem of updating a sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ into two basic processes. Firstly, there is an implicit simple algorithm in the definition of an α-relaxed program that transforms an ELP into a ω-program. Secondly, the resulting ω-program should be computed by DLV in order to check its models, which are called *optimal answer sets*.

By following the notation introduced in [Joh90], the complexity of a DLV program with weak constraints is known to be exponential, bounded by co-NEXPTIME$^{\mathrm{NP}}$ [LPF+06]. The *program transformation*, on the other hand, may be reduced to the problem of *tagging* each rule of the programs in the sequence $(\Pi_1, \Pi_2, \ldots, \Pi_{n-1})$ with $1, 2, \ldots, r$ in the definition of α-relaxed program, where $r = \sum_{i=1}^{n-1} |\Pi_i|$. As a consequence, the following obvious result holds.

Proposition 7.2. *The complexity of transforming any sequence* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *of ELP's into an* α*-relaxed program is linear with respect to* $r = \sum_{i=1}^{n-1} |\Pi_i|$.

Finally, the size of the input to DLV, produced by weakening rules and introducing abducibles, grows *exponentially to the number of rules* in $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$ by having an upper bound of $2^{|\mathcal{A}^*|}$ possible *combinations to include abducibles* into every interpretation —see Example 3.3.

As a result, computing weak constraints and DLP's is very expensive for known to be at the *second level of the polynomial hierarchy* [EFST02]. This is a clear shortcoming for semantics of updates of logic programs in ASP so as to have typical *industrial applications* of, say, *updating large knowledge bases*. Notwithstanding, the implementation to process *toy examples* in the classroom is an immediate practical application, as well as others like a *properties testbed* and more complex prototypes, to mention a few, that shouldn't be despised.

7.6 Discusion

This section has introduced general methods for *rapid prototyping* in DLV of logic programming semantics and for further research in *optimisation techniques*, as well as directions to implement the declarative version of both an update semantics and MSGAS's. The system has been developed with strong emphasis on *declarative programming*, in just some few fragments of imperative-procedural modules, in order to make it easily modifiable for particular frameworks and to confirm claims of the original semantics here presented. Another of its highlights is its modularity and *UNIX paradigm* that allows it to be a *web service* and easily plugged into other systems via on-line even without needing to download it. Moreover, its simple standard *graphical user interface* in HTML makes it very easy to use, compared to most of the solvers implemented for command-line use.

As one of the main components of Logic Programming, implementation of semantics helps quickly understand it (for educational proposes and for a reliable comparison tool, for instance), spread it, test properties and compute knowledge bases for more complex prototypes and other frameworks. In addition, an analysis on the *complexity* of the prototype shows that the transformation from a given sequence of logic programs is polynomial, while computing the resulting transformed program may be exponential, due to DLV's weak constraints. Nevertheless, the implementation shouldn't be despised, as it has immediate practical academic applications at least.

8 Conclusions

This paper is a proposal of a semantics for updates that performs the methods presented in previous reports of relaxing an original knowledge base and of establishing preferences amongst candidate models. In particular, the intuition behind the methodology consists in favouring the latest updates that conflict with an original knowledge base by means of *preferring generalised answer sets*, and also by preserving *consistency*. In addition, it emphasises the importance of an approach based on key *structural properties*, and it may be used for EDLP's.

The core of this paper shows that the new semantics satisfies the introduced set of *structural properties for sequences* of updating logic programs, as well as other needed properties of *consistency preservation* and *inconsistent updates*. The latter set of properties has to do with *consistency restoration* both from an original inconsistent knowledge base and an inconsistent observation, which makes the *difference between belief revision and updates* a little fuzzy. Nevertheless, such a difference is secondary in this research, as the general goal is to produce a robust semantics to represent knowledge, with intuitive behaviour.

As a result, the paper consists of two alternatives to restore consistency. Firstly, a null update to an inconsistent knowledge base shall relax the source of inconsistency. Secondly, a slight modification to the semantics so that it relaxes the entire sequence including the update.

Besides satisfying the principles proposed, this chapter illustrates through several examples and transformations how to overcome problems occurring in alternative update approaches for ASP. One of the key transformations yields an abductive program out of a given sequence of updating extended logic programs. Another transformation constructs a *preferred weak-constraint program* to find *generalised answer sets* of the abductive program. Finally, as one main goal of Logic Programming, Section 7 includes a *functional prototype* from the declarative semantics version, that finds the update answer sets of a given sequence by means of DLV's *weak-constraints models*. The prototype is fully functional and runs online with a standard browser interface. The section also includes an analysis of complexity of the corresponding processes.

9 Acknowledgement

I am very grateful to Wojciech Jamroga for his discussions and many useful comments on this work.

References

[ABBL04] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. Semantics for dynamic logic programming: A principle-based approach. In V. Lifschitz and I. Niemelä, editors, *LPNMR-7*, LNCS, pages 8–20, Fort Lauderdale, FL, USA, 2004. Springer.

[ABBL05] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.

[ALP+99] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1–3):43–70, 1999.

[BG03] Marcello Balduccini and Michael Gelfond. Logic programs with consistency-restoring rules. In *Proceedings of the AAAI Spring 2003 Symposium*, pages 9–18, Palo Alto, California, 2003. AAAI Press.

[Bre02] Gerhard Brewka. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002*, Edmonton, Alberta, Canada, 2002. Morgan Kaufmann.

[Dix95] Jürgen Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[EFST00] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Considerations on updates of logic programs. In Manuel Ojeda-Aciego, Inman P. de Guzmán, Gerhard Brewka, and L. Moniz Pereira, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000*, pages 2–20, Malaga, Spain, 2000. Springer Verlag.

[EFST02] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6):711–767, 2002.

[EFST05] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Reasoning about evolving nonmonotonic knowledge bases. *ACM Transactions on Computational Logic*, 6(2):389–440, 2005.

[GDO06] Juan C. Guadarrama, Jürgen Dix, and Mauricio Osorio. Update sequences in Generalised Answer Set Programming based on structural properties. In Patrick Kellenberger, editor, *Special Session of the 5th International MICAI Conference*, pages 32–41, Mexico City, Mexico, 2006. IEEE Computer Society. ISBN: 0-7695-2722-1.

[GDOZ05] Juan C. Guadarrama, Jürgen Dix, Mauricio Osorio, and Fernando Zacarías. Updates in Answer Set Programming based on structural properties. In Sheila McIlraith, Pavlos Peppas, and Michael Thielscher, editors, *7th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 213–219, Corfu, Greece, May 2005. Fakultät Informatik, ISSN 1430-211X.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium ICLP/SLP*, Seattle, Washington, 1988. MIT Press.

[Gua07a] Juan C. Guadarrama. Implementing knowledge update sequences. In Alexander Gelbukh and Angel Kuri Morales, editors, *MICAI 2007: Advances in Artificial Intelligence*, volume 4827 of *LNCS*, pages 260–270, Aguascalientes, Mexico, November 2007. Springer-Verlag.

[Gua07b] Juan C. Guadarrama. Maintaining knowledge bases at the object level. In Alexander Gelbukh and Angel F. Kuri Morales, editors, *Special Session of the 6th International MICAI Conference*, pages 3–13, Aguascalientes, Mexico, November 2007. IEEE Computer Society. ISBN: 978-0-7695-3124-3.

[Gua07c] Juan C. Guadarrama. A road map of updating in ASP. ISSN: 1860-8477 IfI-07-16, Institute für Informatik, TU-Clausthal, Clausthal, Germany, December 2007.

[Gua08] Juan C. Guadarrama. Update operation in ASP revisited. ISSN: 1860-8477 IfI-08-12, Institute für Informatik, TU-Clausthal, Clausthal, Germany, December 2008.

[IS04] Katsumi Inoue and Chiaki Sakama. Equivalence of logic programs under updates. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, volume 3229 of *LNAI*, pages 174–186, Lisbon, Portugal, 2004. Springer-Verlag.

[Joh90] David S. Johnson. A catalog of complexity classes. In *Handbook of theoretical computer science: algorithms and complexity*, volume A, pages 67–161. MIT Press, Cambridge, MA, USA, 1990.

[KM89] Hirofumi Katsuno and Alberto O. Mendelzon. A unified view of propositional knowledge base updates. In N. S. Sridharan, editor, *the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 1413–1419, Detroit, Michigan, USA, 1989. Morgan Kaufmann.

[KM90] Antonis C. Kakas and Paolo Mancarella. Generalized Stable Models: A semantics for abduction. In *ECAI*, pages 385–391, Stockholm, Sweden, 1990.

[KM91a] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In *KR'91*, Cambridge, Massachusetts, USA, 1991. Morgan Kaufmann Publishers.

[KM91b] Hirofumi Katsuno and Alberto O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1991.

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[LPV01] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, October 2001.

[Nel49] David Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14(1):16–26, 1949.

[NS97] Ilkka Niemela and Patrik Simons. Smodels —an implementation of the Stable Model and Well-Founded Semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence (LNCS)*, pages 420–429, Dagstuhl Castle, Germany, 1997. Springer.

[OC07] Mauricio Osorio and Victor Cuevas. Updates in answer set programming: An approach based on basic structural properties. *Journal of Theory and Practice of Logic Programming*, 7(4):451–479, 2007.

[ONA01] Mauricio Osorio, Juan Antonio Navarro, and José Arrazola. Equivalence in answer set programming. In Alberto Pettorossi, editor, *Logic Based Program Synthesis and Transformation:* 11^{th} *International Workshop; selected papers / LOPSTR 2001, Paphos, Cyprus*, LNCS 2372, pages 57–75, Paphos, Cyprus, November 2001. Springer-Verlag Berlin.

[OZ04] Mauricio Osorio and Fernando Zacarías. On updates of logic programs: A properties-based approach. In *FoIKS*, pages 231–241, Wilhelminenburg Castle, Austria, 2004. Springer.

[Pea97] David Pearce. A new logical characterisation of stable models and answer sets. In *NMELP '96: Selected papers from the Non-Monotonic Extensions of Logic Programming*, volume 1216/1997 of *LNCS*, pages 57–70, London, UK, 1997. Springer Berlin/Heidelberg.

[Pea99a] David Pearce. From here to there: Stable negation in logic programming. In Dov M. Gabbay and Heinrich Wansing, editors, *What Is Negation?*, volume 13 of *Applied Logic Series*, pages 161–181. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[Pea99b] David Pearce. Stable inference as intuitionistic validity. *Logic Programming*, 38:79–91, 1999.

[SI03] Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming*, 3(6):671–715, 2003.

[ZF05] Yan Zhang and Norman Foo. A unified framework for representing logic program updates. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 707–713, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press / The MIT Press.